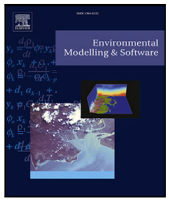
Environmental Modelling and Software 160 (2023) 105600

Contents lists available at ScienceDirect

Environmental Modelling and Software

journal homepage: www.elsevier.com/locate/envsoft



Position Paper

PLUME Dashboard: A free and open-source mobile air quality monitoring dashboard

Chris Kelly [a], Julian Fawkes [a], Rachel Habermehl [a], Davi de Ferreyro Monticelli [b], Naomi Zimmerman [a,b,*]

[a] Department of Mechanical Engineering, 2054-6250 Applied Science Lane, Vancouver, V6T 1Z4, British Columbia, Canada
[b] Department of Earth, Ocean, and Atmospheric Sciences, 2207 Main Mall, Vancouver, V6T 1Z4, British Columbia, Canada

ARTICLE INFO

Keywords:
Mobile monitoring
Air pollution
Peak detection
Open-source Python script
Data interpretation
Data visualization

ABSTRACT

The deployment of a mobile air quality monitoring laboratory requires advanced real-time instrument monitoring and data management software, which can be prohibitively expensive. In this work we present the PLUME Dashboard: a software package built in Python designed specifically for mobile air quality monitoring purposes. It aims to provide a free and open-source alternative to comparable commercial packages, thus reducing the barrier to entry of conducting such research. This paper outlines the development of the PLUME Dashboard and justifies the design choices that were made while also providing thorough documentation and explanation for how the software works. Functionality includes real-time data display, real-time peak identification, baseline subtraction, real-time air quality and self-sampling alerts (based on wind direction and vehicle speed), and post-processing tools such as peak identification and map merging with GPS data. The functionality of PLUME Dashboard is tested using real-world data collected in Toronto and Vancouver Canada.

Software availability

PLUME Dashboard version 1.0.0 was developed by UBC's iREACH research group [https://ireach.mech.ubc.ca]. It is a free and open-source program which can be downloaded via the GitHub repository [https://github.com/iREACH-UBC/PLUME_Dashboard]. PLUME Dashboard version 1.0 was initially released in 2022 under General Public License v3.0 (GPL-3.0) and was developed to run on Windows; the authors have not tested it on Macintosh or Linux operating systems. It was written entirely in Python and will work with version ≥ 3.7. PLUME Dashboard's program files are less than 150 kB. In order to run PLUME Dashboard, the user will need to install three pieces of downloadable software (JET BRAINS, 2022; Van Rossum and Drake, 2009; Campailla and Giordani, 2022) and various Python packages (Evans, 2022; Hossain, 2019; Harris et al., 2020; Redis Inc., 2022; The pandas development team, 2020; Plotly Technologies Inc., 2015, 2022; Nokeri, 2022; Parmer, 2022; Gazoni and Clark, 2022; Collins, 2022; Lefebvre, 2022). Refer to the Supporting Information or the GitHub repository's README.md for a comprehensive list of these requirements and an installation guide.

1. Introduction

The adverse effects of poor air quality is a growing concern (de Medrano et al., 2021). Air pollution is known to be one of the leading causes of premature mortality worldwide (Badyda et al., 2016) and has been linked to the prevalence of various respiratory diseases, such as asthma (Ho et al., 2007). In major urban centers, air pollution concentrations can exhibit extreme variations both spatially and temporally (Alavi-Shoshtari et al., 2018). Major transportation centers and corridors in particular (e.g. large intersections, highways) can exhibit highly elevated air pollutant concentrations compared to typical background levels (Deshmukh et al., 2020). In addition, research suggests that to capture the effect of emission plumes, and short-lived peaks of atmospheric pollution from local sources such as traffic and natural gas production, there is a need for shorter time resolution than what is available in most stationary monitoring facilities (Brantley et al., 2014). This is especially true for certain pollutants such as ultrafine particles (UFP) (Ketzel and Berkowicz, 2004), methane (Xia et al., 2022), and odors (i.e., chemical mix that cause the smell) (Bax et al., 2021). In other words, detecting emissions from fugitive sources, leaks and industrial malfunction require instrumentation with high time resolution. Given the possible interaction of UFP, methane, and odors with other co-emitted standard criteria air contaminants (e.g., CO, CO2, NOx) it is also valuable to measure the latter in real-time to aid in source identification. As a result, having high spatial and temporal monitoring resolution is necessary to accurately measure air pollution

* Corresponding author at: Department of Mechanical Engineering, 2054-6250 Applied Science Lane, Vancouver, V6T 1Z4, British Columbia, Canada.
E-mail address: nzimmerman@mech.ubc.ca (N. Zimmerman).

https://doi.org/10.1016/j.envsoft.2022.105600
Received 31 July 2022; Received in revised form 20 November 2022; Accepted 1 December 2022
Available online 3 December 2022

in urban settings. Consequently, being able to reliably collect and process real-time air quality data is of critical importance.

Traditional air pollution monitoring stations do not lend themselves well to meeting these demands, due to both financial and practical limitations (Hofman et al., 2022). These stations tend to be sparsely located and typically provide time resolution on an hourly or once per minute basis at best. An effective solution to address these problems is the deployment of mobile air quality monitoring laboratories (Bukowiecki et al., 2002). This is typically achieved by outfitting a large vehicle with an air sampling inlet, onboard power supply, high time-resolution air quality monitoring instruments, and the necessary supporting software and data management systems. Once set up, air quality measurements can be made while the vehicle is in motion, thus allowing for enhanced spatial and temporal measurement resolution (i.e., the capacity to take new measurements in short time intervals). This approach has been used to measure on-road air pollution (Wang et al., 2009; Canagaratna et al., 2004) as well as ambient pollution concentrations in a range of environments (Maciejczyk et al., 2004; Herndon et al., 2005), among other applications.

A key non-hardware requirement of a mobile laboratory is the supporting software and data management systems, which present two main problems. Firstly, the fast paced and variable nature of measurements in an urban setting necessitates the use of real-time data visualization software connected to all onboard instruments. This would provide the user with a constantly updating overview of the current air pollution levels. Unfortunately, many modern instrument monitoring and visualization software packages are prohibitively expensive. Secondly, the presence of multiple instruments, each with a very high time measurement resolution, can result in very large datasets. Thus, processing and analyzing the collected data in an efficient and streamlined manner can be challenging (Drewnick et al., 2012). While it is true that low-cost sensors have been developed and applied for environmental measurements at high spatial resolution, they still face the issue of low accuracy, lower time resolution (typically 1 min or longer) and often need advanced calibration routines and algorithms to be reliable (Zimmerman, 2022). Likewise, open-source packages designed to improve air quality data analysis are available (Feenstra et al., 2020; Mahajan, 2022). However, to the best of our knowledge, they are scarce for mobile monitoring applications, or are restricted to the use of low-cost sensors with wireless communication (e.g., Vagnoli et al. (2014)) and the number of pollutants measured (e.g., Lim et al. (2019) and Khedo et al. (2020)). Furthermore, existing open-source software does not provide the real-time features important for mobile monitoring (e.g., threshold alerts, peak detection, wind direction alerts) or the capability to work with simulated or historical data.

The objective of this work is to provide a Python-based software package that addresses these two problems. PLUME (Portable Laboratory for Understanding human-Made Emissions) Dashboard is easy to integrate and provides the user with real-time data visualization for all connected instruments. Furthermore, it is free and open-source, therefore improving financial accessibility of modern instrument monitoring and visualization software while still maintaining the essential functionality of such software. In addition to this, PLUME Dashboard comes equipped with a wide array of data processing aids, hence streamlining the process of analyzing data collected from mobile laboratories. PLUME Dashboard is intended for researchers, specifically those who are looking to deploy a mobile monitoring laboratory but are not able to afford (or would be financially burdened by) more expensive data monitoring software. Additionally, since the scripts behind the PLUME Dashboard are all fully available, this software provides the possibility for enhanced customization at no cost. This paper summarizes the functionality of PLUME Dashboard as well as provides a practical example of how it is used based on data collected in Toronto, Ontario and Vancouver, British Columbia, Canada.

2. Methods

2.1. Dashboard overview

2.1.1. Script layout

PLUME Dashboard is composed of several Python (Van Rossum and Drake, 2009) scripts. At the core of the dashboard is the main script, which handles the operation of the dashboard as well as generating and updating the live plots. The dashboard uses several user-defined settings, whose values can be set by modifying the "user_defined_settings.ini" file or via temporary changes while the dashboard is running using various commands. If the user accidentally deletes or incorrectly formats the "user_defined_settings.ini" file, it can be generated again (with the default settings) by running the "create_default_config" script. For the full documentation of the user-defined settings, see Section S2 in the Supporting Information. In addition to the main script, operation of the dashboard also requires at least one data acquisition (DAQ) script to be running, which is responsible for passing data to the main script via a locally-hosted Redis server (Campailla and Giordani, 2022); it is recommended to use a machine with a minimum of 4 GB of RAM available. A Redis server will store data in the computer's memory such that it can be accessed by any script running on the computer. Details of setting up the Redis server are provided in the Supporting Information as well as the GitHub Repository README.md. The dashboard comes with a pre-written Modbus-TCP DAQ script, which can pull data from any Modbus-TCP capable datalogger or instrument and pass it to the dashboard. Non-Modbus-TCP capable instruments will need to have a separate DAQ script written for them. A template for writing such a script is provided in this paper. In the Section S6 in Supporting Information we provide a discussion of the PLUME Dashboard's capacity to process multiple instruments. Lastly, there is a baseline subtraction script, post processing peak detection script, and GPS data merging script, which can all be run after data has been collected and recorded by the dashboard.

2.1.2. Main script

The main script will receive data from the DAQ scripts and update the live plot as new data is received. To do this, the main script appends the incoming data for each pollutant as well as the current time to separate Python deque (doubly ended queue) objects, which are essentially Python lists of fixed maximum length. The live plot is then updated to display the time deque on the x-axis and the pollutant's respective deque on the y-axis for each of the pollutants the user has selected to be displayed (with a maximum of four pollutants that can be displayed at once). The y-range of each graph is fixed and can be uniquely set for each pollutant before running the dashboard (via the "user_defined_settings.ini" file) or temporarily changed while the dashboard is running. The user can also choose to enable the "autoscale" feature for each pollutant, which will automatically scale the y-ranges of the live plots based on the current minimum and maximum points. Using Python deques as opposed to lists is an advantageous approach since Python will automatically delete the leftmost data and shift the items to the left to accommodate for the new incoming data once the maximum length is reached. PLUME Dashboard also includes a simulated data feature, which can be used to run existing datasets through the dashboard for testing purposes (outlined in detail in Section 2.3). If enabled, the incoming pollutant data from Redis will be overridden with the simulated data.

2.1.3. DAQ scripts

The DAQ scripts are responsible for passing data to the main script via a locally-hosted Redis server (Redis Inc., 2022; Campailla and Giordani, 2022). Ideally, one would want all their instruments to be compatible with a data logger so that one DAQ script could be used for all of the instruments. To avoid complications the user should opt for a data logger capable of sending data over a Modbus-TCP connection.

Fig. 1. Screenshot of PLUME Dashboard with main components labeled.

If so, the MODBUS-TCP_DAQ script can be used without needing to be modified. All data must be sent as a 32-bit float or long (no reversal of byte order) over Modbus address 1. The other connection parameters (i.e., port and IP address) as well as which Modbus holding registers to be read for each pollutant can be set in the "user_defined_settings.ini" file. If it is unknown which holding registers each pollutant's data is stored in, Modbus scanning software can be used to assist the user; we used the program "CASModbusScanner" (Chipkin Automation Systems, 2022). If one does not wish to use a Modbus capable data logger/device, then a new DAQ script will need to be written. A template for writing such a DAQ script is provided in the Supporting Information, Section S5.

Using separate standalone DAQ scripts to pass data to the dashboard as opposed to running it through a single main script provides two main benefits. Firstly, the code in the main script can be simplified as it can bypass multithreading and multiprocessing limitations that would otherwise be present if the entire program was in the main script. Secondly, it allows for new DAQ scripts to be easily written using just a few lines of code by following a basic template (provided in the Supporting Information, Section S5), thus simplifying the process of integrating new non-Modbus capable instruments/devices into the dashboard. One minor drawback to this approach is that starting up the dashboard will require the user to run Redis as well as multiple Python scripts, thus being more complicated than simply launching a single program.

2.1.4. Output files

While in operation, the dashboard creates and writes to two separate comma-separated value (CSV) files: the Event Markers CSV file (formatted as "Event Markers yyyy-mm-dd.csv") and the Sensor Transcript CSV file (formatted as "Sensor Transcript yyyy-mm-dd.csv"). A corresponding backup text (.txt) file is also created for each CSV file. The Sensor Transcript CSV file is simply a log of the entire dataset with columns for the pollutant values and time. Whenever the live plot updates, the current time and pollutant values are added as a row to the Sensor Transcript CSV. The benefits of implementing this Sensor Transcript CSV are two-fold. First off, it enables the data to be easily analyzed and processed by a wide range of programs (e.g., one can run the OpenAir R package Carslaw and Ropkins, 2012 after changing the 'Time' column for 'date'). Additionally, Sensor Transcript CSV files are formatted such that they can be directly fed into the included baseline subtraction and post-processing peak detection scripts after the data has been collected.

The Event Markers CSV file contains the potential events of interest detected by the dashboard using its various real-time data processing aids and algorithms. Similarly to the Sensor Transcript CSV, the Event Markers CSV has columns for the time and pollutant values. In addition to this, it also has columns for the type of event detected, the pollutant that triggered the event, and the name (or "event tag") of the event. A detailed explanation of each CSV column is provided in Section S3 of the Supporting Information. Whenever a potential event of interest is detected, the current time, pollutant values, and other previously described column values will be added as a row to the Event Markers CSV file. In addition to automatically-detected events being recorded, the user can manually trigger an event to be recorded whenever a visible event of interest occurs (e.g., a car driving by the mobile laboratory). To do this, the user must input an event name (which will populate the "event tag" column) into the text entry box on the dashboard display and click the "Mark Event" button. The user can also enter various "commands" into the text entry box, which are outlined in Section 2.1.5.

2.1.5. Event marker "commands" and overall layout

Some user-defined settings can be changed individually for each pollutant in real-time while the dashboard is operating. This is done by inputting various commands into the text entry box and clicking the "Mark Event" button; essentially the same procedure as creating a manually inputted event of interest. All commands must begin with the "*" character to indicate to the program that the following text will be a command (see Supporting Information for syntax examples). The settings that can be changed via commands include, *A1_coeff*, the *n* percentile used in Algorithm 1 (A1), the *AQ warning threshold*, the range used by wind direction range warning, the y-ranges of the live graphs, and the autoscale feature for each pollutant. Additionally, A1, the AQ warning algorithm, and the wind direction range warning can be toggled ON or OFF with a single command (see Supporting Information Section S2 for full documentation). The purpose of the commands feature is to enable the user to change settings on the fly in order to adapt to quickly changing conditions, without needing to restart the dashboard.

Only four timeseries plots (plus the wind rose and pollutants bars) can be displayed simultaneously. Each timeseries plot will show 60 s

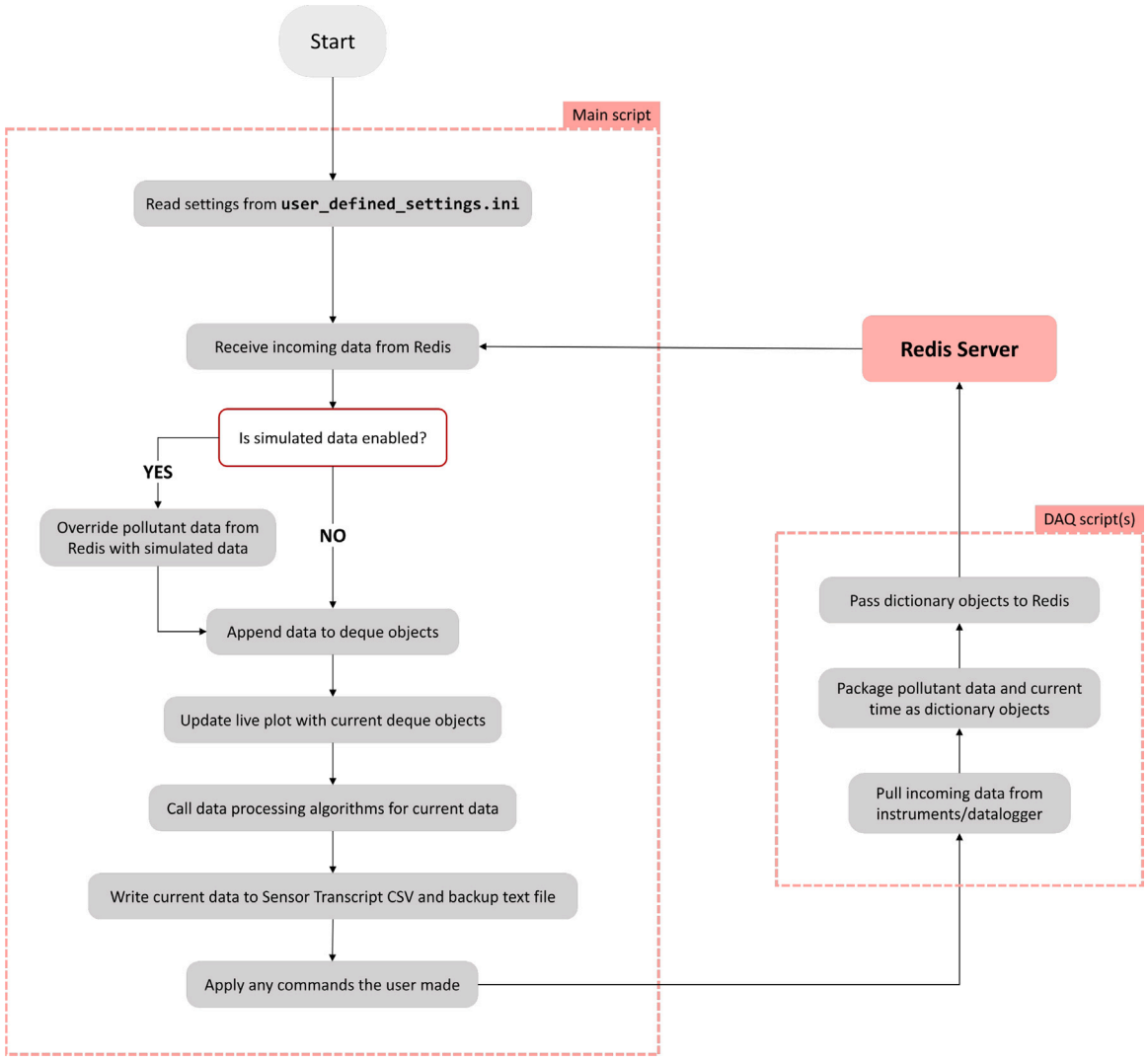


Fig. 2. Flowchart summarizing the general operation of the dashboard.

of data (30 s is just the recommended start up bypass time for peak detection, which is unrelated to the deque size). Despite this four-plot maximum, data processing (i.e., peak detection, AQ over/under warning, etc.) will always take place for all pollutants at all times, regardless of which are currently being displayed as a timeseries. Fig. 1 is a screenshot of PLUME Dashboard's user interface with each major component being marked and labeled. A flow chart is provided in Fig. 2 which summarizes the general operation of the dashboard.

2.2. Data processing aids

2.2.1. Real-time peak detection algorithm

When collecting high frequency air quality data over the span of several days in multiple locations, there will be hundreds of thousands of data points for each pollutant, which creates challenges for interpretation. Recent research has focused on methods to bootstrap these large datasets to facilitate posterior spatial analysis (Chen et al., 2022). Temporally, real-time peak detection can reduce analysis complexity by automatically flagging events (i.e., "peaks" in the dataset) that may be of interest, allowing researchers to focus on time periods of interest. The dashboard implements an automatic peak detection algorithm (referred to as "A1" in the dashboard), which uses a modified version of a method described by Drewnick et al. (2012). Its purpose is to semi-automate data processing by detecting peaks in real-time and recording the event to the Event Markers CSV file. Every time a new data point is collected and displayed on the dashboard, A1 will calculate a new threshold value, t (Eq. (1)), which represents the minimum value that the point must reach to be considered a peak. The algorithm still requires judgment from the user in terms of the parameters used to detect peaks, since the imprecise choice of parameters can potentially result in peaks going undetected, however, we suggest that researchers use a short period of data to tune their parameter choice based on their analysis goals. Once optimized, these can be quickly applied to the entirety of the dataset.

$$t = sn \times A1_{coeff} + bm \tag{1}$$

In Eq. (1), sn is the standard deviation (s) of all on-screen data points that fall under the nth percentile (n), which was set to 50% in this study as that is what was recommended by Drewnick et al. (2012), bm is the mth percentile (m) of all on-screen data points, and $A1\_coeff$ is a sensitivity coefficient. Theoretically, $A1\_coeff$ can be set to any positive integer. Running the post-processing peak detection script, upon visual inspection, we found that setting it to 15 produced satisfactory results in most cases. If one finds that peaks are being falsely detected, then it is advised to increase the value of $A1\_coeff$ until desirable results are achieved. Likewise, if one finds that peaks are going undetected, then the solution would be to decrease the value of $A1\_coeff$ until the undetected peaks in question are being detected. n, m, and $A1\_coeff$ are

all user-defined settings whose values can be set individually for each pollutant in the "user_defined_settings.ini" file.

The purpose of adding the *bm* term is to increase the value of t to the expected minimum value of that pollutant, thus it is recommended that it be set to a very low percentile, such as 1%. Using a low percentile as opposed to the absolute minimum value is beneficial because it will not be affected by extraneously low values caused by instrument errors. If the user wishes to disable the *bm* term, setting m to 0% will result in the dashboard setting the *bm* term to be 0 for that pollutant. If the previous point was marked as a peak, then t will be calculated using a greater threshold. This threshold increase in t is given by (Eqs. (2) and (3)).

$$Inc = \sqrt{c} \times sn \tag{2}$$

$$t = sn \times A1_{coeff} + bm + \sqrt{c} \times sn \tag{3}$$

In Eqs. (2) and (3), c starts at 0 and will be set to 1 once a peak is detected. If the next point immediately after is also marked as a peak, then the value of c will increase by 1. This will continue until a point is reached that is not marked as a peak, at which point c will be set to 0 and the entire process will repeat. The customizability of the n and m percentiles as well as the *A1_coeff* setting allows A1 to be versatile and work well with most datasets and pollutants. It should be noted however that A1 requires the pollutant deques to be reasonably full (at least 30 entries) to correctly detect peaks. Since entries are added at a rate of approximately once per second for 1-s time resolution instruments typical of mobile monitoring, it will take roughly 30 s after launching the dashboard for A1 to work properly. To prevent peaks from falsely being detected during this time, A1 will disable itself until the deques reach a certain size, known as the "startup_bypass", which is a user-defined setting (30 was used for this study). A1 is implemented as a function in the main script. Fig. 3 is a flowchart of the process that the A1 function follows.

2.2.2. Baseline subtraction

Pollutant concentrations measured by a mobile laboratory include signals from both local emission sources of interest (e.g., vehicle exhaust) as well as background sources within the general area (e.g., wildfire smoke). To isolate the contributions from local emission sources of interest (Liu and Zimmerman, 2021) or temporally short-lived (<2 h) pollution events from persistent events (>8 h) (Zimmerman et al., 2020), a pollutant baseline must be generated which represents the background sources within the general area. Subtracting this baseline from the measured pollutant values will effectively isolate the signals or events of interest.

The baseline algorithm (adapted from Wang et al. (2018)) works by finding the local minimum value and performing interpolation in a user-defined time window. We also developed a few extra features to optimize the baseline estimation. One optimization is the capacity of testing multiple values for the parameters used by the algorithm, thus creating more than one baseline approximation for the user to judge its value. For instance, by observing Fig. 5 Case B and D, one may note how in Case B the baseline follows the smaller peaks minima more closely than in Case D (e.g., see timestamp 16:00), while for large peaks the result is virtually the same. Considering the final goal to be identifying 'local' signals and excluding background concentrations, the Case D example may the most appropriate for this dataset.We note, however, that even for Case D there is room for improvement in the algorithm. At the interval 15:55 to 16:10 there is a steady elevation followed by a steady decrease event similar to a entering and leaving an emission plume. During this time, our baseline was also elevated, meaning that subtracting it from the original measurements would result in losing this plume characteristic, and the final data would consist of small and narrow peaks which are likely instrument noises. We had initially developed an algorithm to account for such events (steady increases followed by steady decreases) but it has not yet passed our validation criteria to be included in this manuscript. We plan to implement a steady-state plume background subtraction feature in future versions of the PLUME Dashboard and such updates will be shared through the GitHub repository. As of publication of this paper, one could use the strategy we suggest when entering a plume of contaminants, described in detail in the Supporting Information, Section S8.

The algorithm will read in a dataset generated by the dashboard (such as the Sensor Transcript CSV file) during operation and output a new CSV file containing the original dataset as well as the calculated baseline for each pollutant. The program will split the dataset into chunks, process it one chunk at a time (a chunk size of 3000 rows was used in this study), and then recombine the chunks in the output file. This is a necessary step as entire datasets can be extremely large, thus potentially being problematic to store in a single Python list. In addition to the dataset itself, the algorithm takes in two parameters: *window_size* and *smoothing_index* (both of which can be set in the "user_defined_settings.ini" file). For each data chunk that is fed into the algorithm, it is separated into individual windows (whose size is given by the *window_size* parameter, and correspond to the same time step of data collection, in our case, seconds). The algorithm then determines the minimum value in each window and linearly interpolates between them, saving the result to a Python list. This process is repeated two more times, each time offsetting the windows' starting point by *window_size* divided by three (rounded down regardless of the decimal point) data points. If *smoothing_index* = 1, the three lists are averaged together. If *smoothing_index* >1, the above process is repeated *smoothing_index* times. Each time the process is carried out, it will use an increased window size of *h*window_size* where *h* starts at 1 and will increase by 1 each time the process is repeated until *h* = *smoothing_index*. The final baseline is obtained by averaging the outputs of all the repeated processes (or just using the output from the single process if *smoothing_index* = 1) and then replacing any points whose calculated baseline value is greater than the actual pollutant value at that point with the actual pollutant value. It is recommended to keep the *window_size* setting to three, four, or five, and use the *smoothing_index* setting to control the baseline's noise. Any values of *window_size* significantly above five will result in an excessively flat baseline. Similarly to A1, this baseline calculation algorithm is implemented as a function, which is called for each pollutant each time a data "chunk" is to be processed and returns the calculated baseline. Fig. 4 provides a flowchart to outline this process.

Because of how the baseline algorithm works, it does have one limitation. To compute the baseline of any given point, data both ahead of and behind that specific point will be required. As a result, the baseline values calculated for the points at the very beginning and very end of the dataset will be incorrect (which is evident upon looking at the very front and very back of the calculated baseline). The amount of incorrect points on the extremities of the graph is dictated by the values of the *smoothing_index* and *h*window_size* settings; the higher the values of *smoothing_index* or *h*window_size* are, the more incorrect points there will be. This behavior is also exhibited at the boundaries between chunks. To address this, the "interlace chunks" feature was implemented, which can be enabled in the "user_defined_settings.ini" file. Interlace chunks will correct these incorrect points by computing the baseline twice, each time with offset chunk boundaries, and then stitching together only the correct points from the two calculated baselines. Fig. 5 shows the output of the baseline post-processing algorithm using different configurations. Comparing Case A to Case B we can see the effect of increasing the smoothing_index. The calculated baseline in Case A follows the pollutant concentrations much more closely than the calculated baseline in Case B. Comparing Case B to Case D, we can see that enabling interlace chunks results in a more accurate baseline for higher values of smoothing_index, this is particularly noticeable at data points near 16:00.

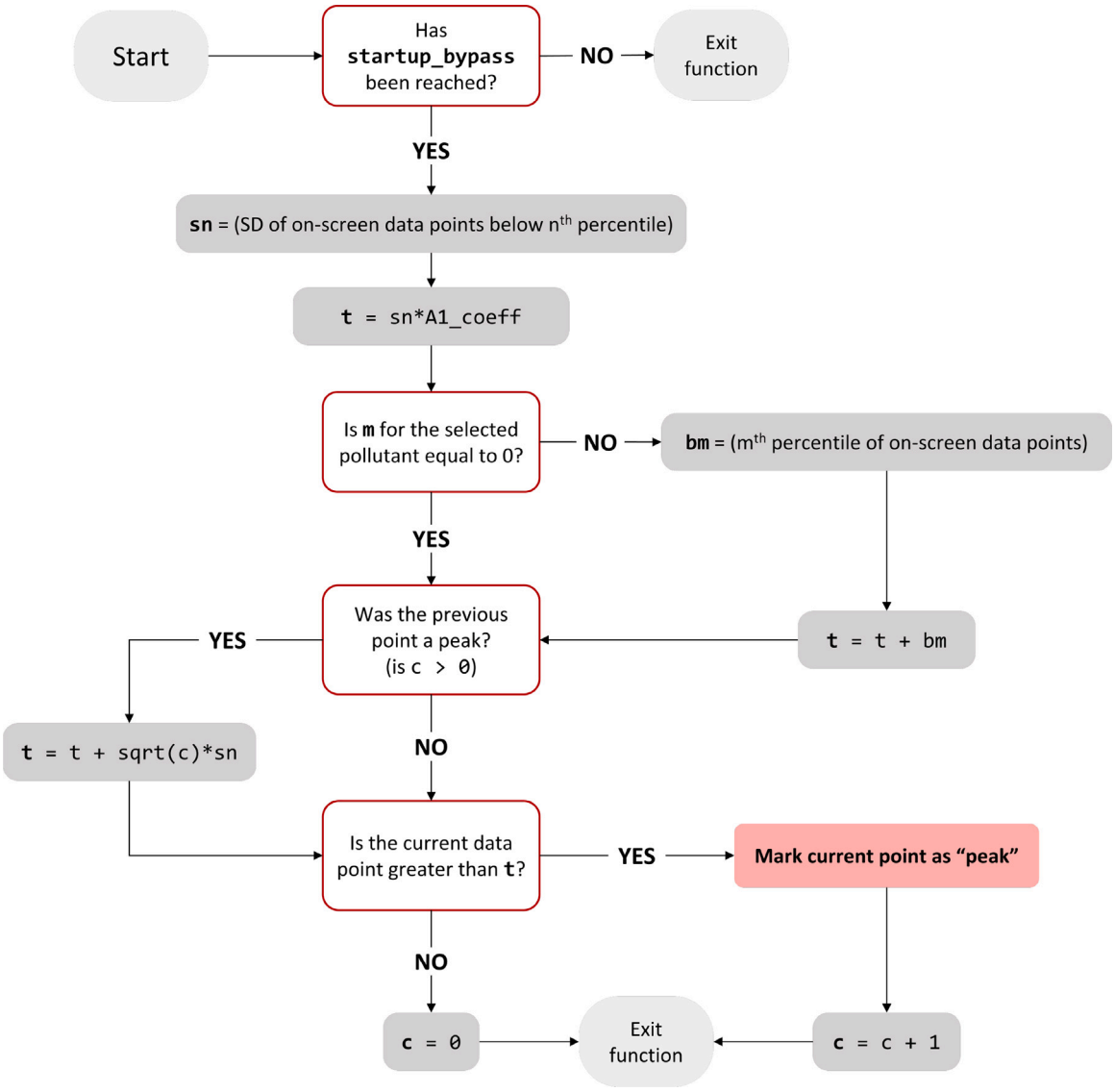


Fig. 3. Flowchart demonstrating operation of A1 function.
Source: Adapted from Drewnick et al. (2012).

2.2.3. Real-time air quality warning event and wind direction range warning

The dashboard also includes a simple air quality (AQ) warning algorithm. This algorithm will detect whenever a pollutant's concentration goes over or under a static user-defined threshold and will record the event to the Event Markers CSV file. The purpose of this algorithm is to detect the start and end of prolonged instances of unacceptably high air pollutant concentrations. The names of the automatically recorded events in Event Markers CSV will indicate whether it is the start or end of the prolonged period of high air pollutant concentration.

Wind direction data uses a slightly modified version of this algorithm, referred to as the "wind direction range warning". It will detect whenever the wind direction enters or leaves a specific user-defined range. This can be useful for determining when the wind is coming from a certain direction of interest (e.g., directly behind the mobile laboratory, which could imply in self-contamination). Using the event marker function of the dashboard is also useful to detect during analysis or log in real-time such events, thus a list of other entries that would facilitate data processing is given in Supporting Information S7.

Currently, the PLUME Dashboard records relative wind direction and there are approaches to use GPS recorded direction plus the sonic anemometer output to determine absolute wind direction. For instance, if the GPS used can provide the traveling direction for each datapoint, one could fix all relative wind direction measurements in a post-processing algorithm by rotating the relative wind direction measured according to the direction traveled, but that is outside the scope of the current dashboard.

2.2.4. Post-processing peak detection

A post-processing peak detection script is available for the user. It can be run on a Sensor Transcript CSV file and will detect peaks in the same way as the real-time algorithm works (Section 2.2.1), calculating a new threshold for each point. Consequently, results from the real-time algorithm can be reproduced in the post-processing peak detection script by using the same settings that were used while the dashboard was in operation. Similarly to how the baseline algorithm works, the post-processing peak detection script will read in a Sensor Transcript CSV, process it one "chunk" at a time, and write the original dataset as well as the results for each pollutant to an output CSV file (with the filename being a user defined setting). Additionally, the user can choose to have the script also write the threshold (t) for each point to the output CSV file. This feature can be enabled or disabled for

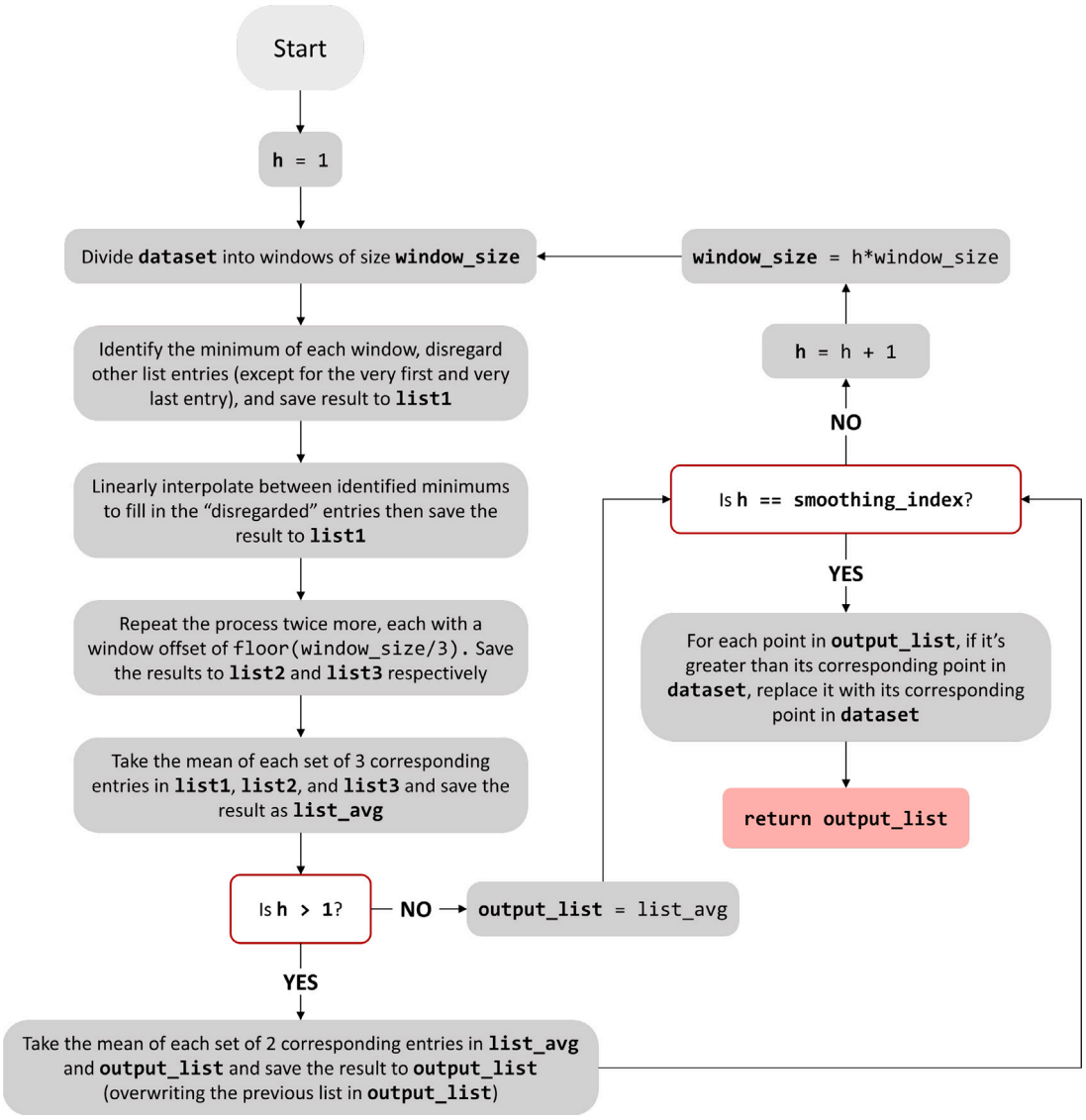


Fig. 4. Flowchart demonstrating operation of baseline function. This is an implementation of the algorithm presented in Wang et al. (2018).

each pollutant individually in the "user_defined_settings.ini" file. The reproducibly of results and ability to see what the threshold is for each point makes the post-processing peak detection script a useful tool for optimizing A1 settings for a particular pollutant.

2.3. Simulated data

For testing purposes, the dashboard comes equipped with a simulated data feature. This feature was primarily implemented to help with testing and validating the dashboard's functionality. It will allow the user to run a dataset through the dashboard as if the data was actually being collected in real-time. Incoming pollutant data from Redis will be overridden, however, time data from Redis will still be used, thus Redis must still be running for this feature to work. This can be set up for each pollutant separately in the "user_defined_settings.ini" file. To do this, the user will need to choose a directory that contains the simulated data spreadsheets, and then specify the filename of each pollutant's simulated data spreadsheet. Simulated data spreadsheets must be either a .csv file or a .xlsx file and should be formatted as an index column and pollutant concentration column. A sample simulated data CSV file is provided in the Github repository linked in the Software Availability section.

2.4. GPS data merging

PLUME Dashboard also comes equipped with a GPS data merging script which will combine sensor transcript data with GPS data (obtained while the dashboard and mobile laboratory are in operation). The script will read in a Sensor Transcript CSV and a GPS Exchange Format (.gpx) file and output a modified Sensor Transcript CSV with additional "latitude" and "longitude" columns from the GPS data. This is accomplished by using the timestamps of the GPS data and sensor transcript data to match a latitude and longitude value to each row in the sensor transcript file (where each row is 1-s data). Because of how mobile air quality monitoring laboratories are designed, sampled air will always take a certain amount of time to reach the measurement instruments (Tan et al., 2014). This amount of time is referred to as "lag time" and is a function of flow rate, pipe/tubing diameter, and overall flow regime inside the sampling line. It can be determined theoretically or by creating a source of high concentration at the intake points and observing how long it takes for instruments to measure a peak. Since it is common to have a separate gas line and a particle line in mobile air quality applications (Wallace et al., 2009), both need to be evaluated. In our case, gas instruments showed a peak after 40 s of exposing the inlet to a high-concentration source, while the particle

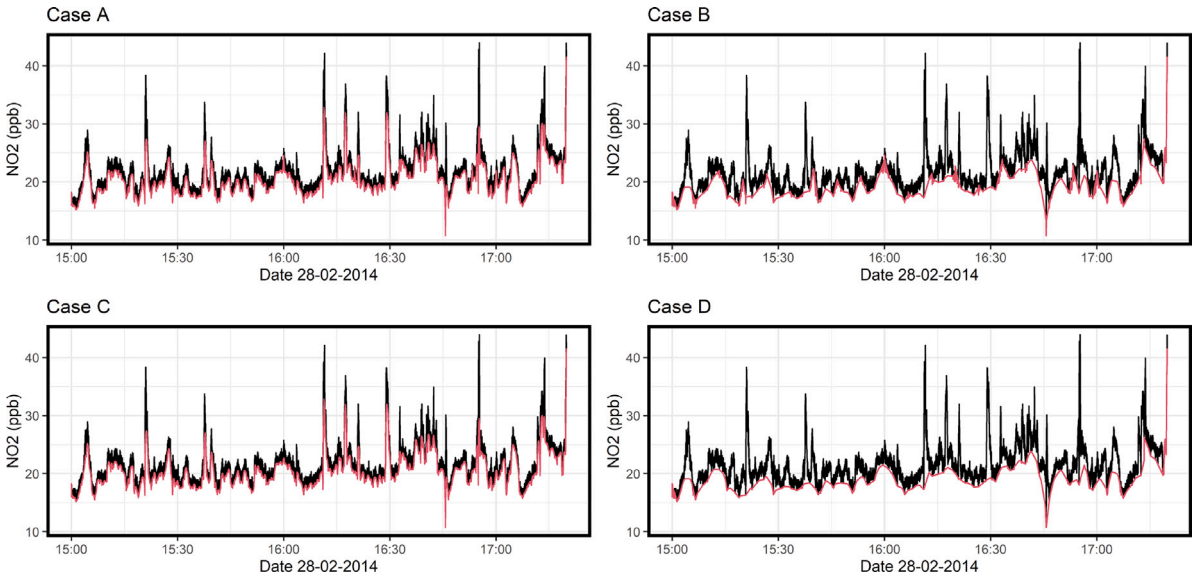


Fig. 5. Baseline (red) post-processing of real data (black) collected by Wang et al. (2015). Case A: smoothing_index = 5, interlace chunks disabled, window size = 3. Case B: smoothing_index = 28, interlace chunks disabled, window size = 3. Case C: smoothing_index = 5, interlace chunks enabled, window size = 3. Case D: smoothing_index = 28, interlace chunks enabled, window size = 3. For all cases, chunk size = 300 (lower than default size of 3000 to highlight effects of interlacing).

instruments took 5 s due to the higher flow rate in the particle line. The lag time for each pollutant can be specified as a user-defined setting and will automatically be taken into account by the GPS data merging script. Before running the GPS data merging script, the user will need to specify the start and end times of the Sensor Transcript CSV as user-defined settings (see Table S3 in Supporting Information for more details). We developed this script using two sets of GPSs, the Garmin Drive™52 (Garmin) and the GL-770 Professional GNSS (Transystem Inc.). There could be minor differences between the GPS Exchange Format from the user GPS and those two.

2.5. Test data acquisition

All the dashboard's data processing aids and features were tested using real data collected in May of 2014 in Toronto, Canada as part of a real-world air quality monitoring study. These studies are described in detail in previous publications (Wang et al., 2015; Zimmerman et al., 2016). Since this data was collected on-road in an urban setting at high time resolution (1 s) it serves as suitable test data that can be used to validate the dashboard's functionality. To test the real-time functionality of the dashboard, the data was run through using the simulated data feature (Section 2.3). Because the dataset contains several hours worth of data, only certain portions corresponding to events of interest were run through the dashboard to highlight dashboard functionality. This included real-time simulated peak detection for the Toronto's dataset (Fig. 6 Middle). We opted to use data from past fieldwork to validate the 'Simulated data' option (Section 2.3) and post-processing features. This way, we demonstrate that previous research could also be processed using our software.

Real-time features of the PLUME Dashboard were also tested during Vancouver-based fieldwork conducted July–August of 2022 (i.e., no simulated data). Here we opt to show the wind direction range warning and the AQ threshold alert as examples (Fig. 6 Top and Bottom). To test the GPS data merging script, we drove our mobile laboratory around the City of Delta (south of Vancouver) in British Columbia, Canada, leaving from and returning to the University of British Columbia (UBC) Vancouver campus. From this run we obtained a sensor transcript CSV and a .gpx GPS file (GPS used: Transystem GL-770). The GPS data merging script was then run using the results of this pilot sampling campaign to obtain the time series and air quality map for CO_2 (as measured by the LICOR 840A), as an example given its spatial variation. Fig. 7 displays the results.

3. Results and discussion

Fig. 6 illustrates that the dashboard was able to successfully detect events of interest (i.e., peaks and AQ over/under events), log said events to the Event Markers CSV file, and change various settings via commands, validating the real-time functionality of the dashboard. To test the post-processing aids of the dashboard, the entire dataset was run through both the baseline subtraction script and the post-processing peak detection script. The results were previously shown in Figs. 5 for the baseline subtraction and 6 for the post-processing peak detection script. In Section S8 of the Supporting Information we demonstrate the effects of toggling with the default parameters of the peak detection algorithm and discuss a best-practice for when the mobile laboratory is inside a plume for prolonged periods (>30 s). We recognize that under its current version, the PLUME Dashboard does not detect the start and end time of a peak in real-time or through post-processing. We will work to develop the next version of our algorithm based on the work of Weller et al. (2019) so it can perform such task in real-time. The operation rationale will be as it follows: when first peak is identified, the algorithm will note the timestamp immediately before as plume start. If there are subsequent peaks <5 s apart, they will be lumped together in one plume. Once time between peaks >5 s, the algorithm will mark data point right after the last peak as end point of plume.

Fig. 5 shows that the baseline subtraction script can compute a multitude of appropriate baselines with various amounts of smoothing while still maintaining its accuracy near chunk boundaries by utilizing the interlace chunks feature. Additionally, Fig. 6 demonstrates that the post-processing peak detection script can successfully detect peaks and reproduce the results from real-time peak detection.

Fig. 7 shows that the script was able to successfully match latitude and longitude values to each row of pollutant data in the sensor transcript file, which in turn can be used to produce geo-referenced air quality maps. Based on our testing of the functionality of the peak detection, air quality threshold alerting, wind direction range alerting, baseline calculation and GPS-air quality map merging on real-data collected in Toronto and Vancouver, Canada, we demonstrate that features of the PLUME Dashboard are operating as intended.

4. Conclusions

With its superior spatial and temporal resolution (compared to a reference station), mobile air quality monitoring is a popular choice for

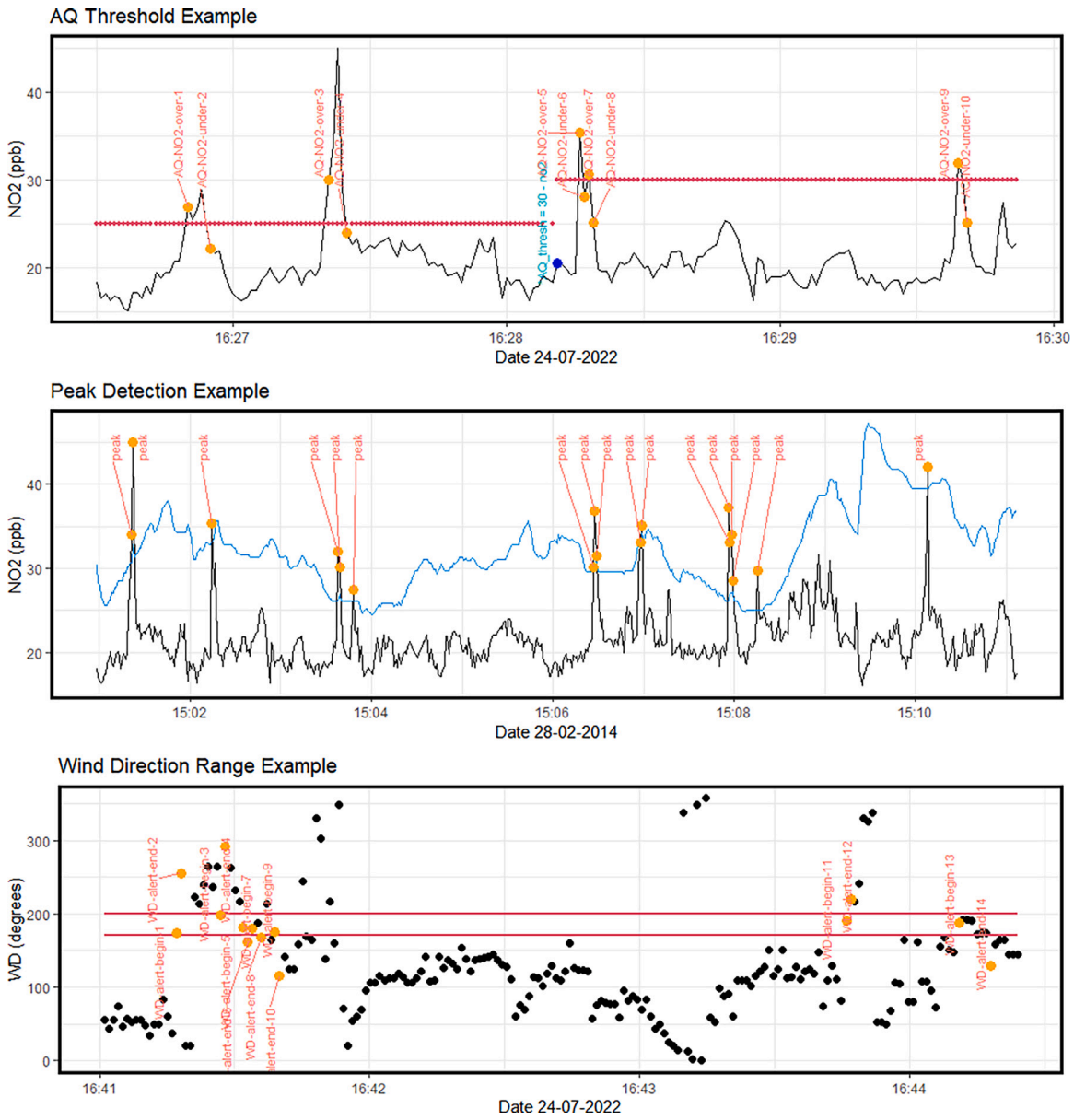


Fig. 6. Top: AQ Threshold Example shows a user change mid-way sampling from an AQ_NO2 alert threshold of 25 ppb to 30 ppb. Middle: Peak Detection Example displays the post-processing algorithm detecting peaks (orange dots) according to pollutant value (black line) and threshold (blue line); in this example A1_Coeff = 15, m = 1, n = 50. Bottom: Wind Direction Range Example flags data points when the wind direction enters and exits the user-defined alerting range of 170–200 degrees; wind direction flag ranges are a user-defined setting.

carrying out urban air quality monitoring by regulatory agencies and academic researchers. However, deploying a mobile air quality monitoring laboratory presents a myriad of unique challenges. Setting up the necessary supporting software and data management systems can prove to be the most problematic task as suitable commercially-available software packages can be expensive and lack customization or data processing features appropriate for air pollution researchers. Furthermore, the datasets obtained from mobile monitoring can be extremely large and therefore cumbersome to process and analyze. PLUME Dashboard addresses these concerns by providing the user with the necessary features to deploy a mobile air quality laboratory while still being free and open-source. PLUME Dashboard is available for download on Github [https://github.com/iREACH-UBC/PLUME_Dashboard] and can be easily connected to any Modbus-capable instruments. As a result, PLUME Dashboard can be a viable option for any research team that has been intending to deploy a mobile monitoring laboratory but has been restrained by financial limitations. Possible future enhancements to PLUME Dashboard may include greater compatibility with non-Modbus capable instruments.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Data availability

All data and code is available in the linked Github repository in the paper.

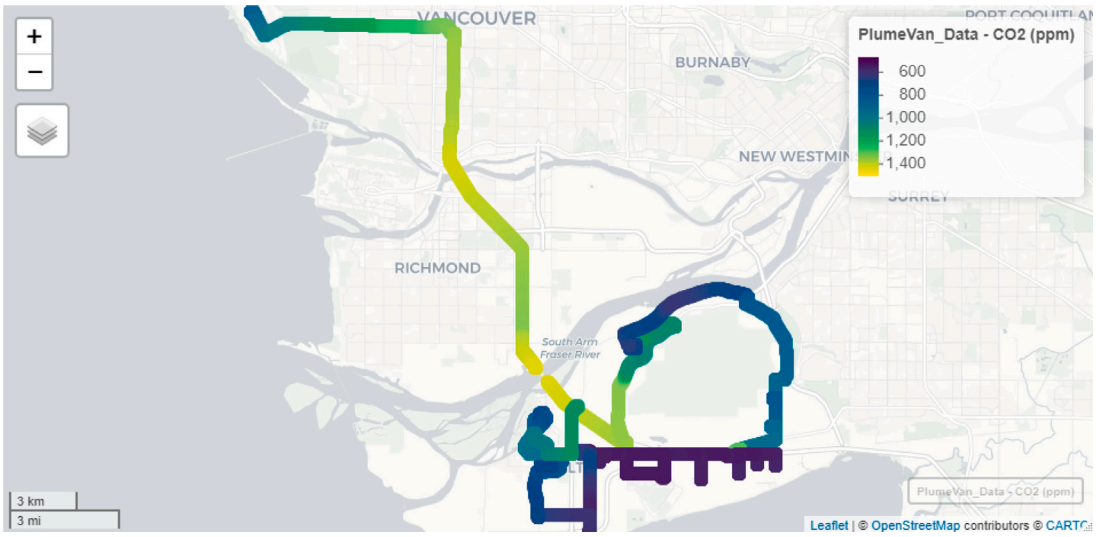


Fig. 7. Map of plotted concentration of CO_2 given the location of the mobile monitoring.

Acknowledgments

This research was undertaken, in part, thanks to funding from the Canada Research Chairs Program. Funding for the PLUME Dashboard was provided by the Natural Sciences and Engineering Research Council (NSERC) Discovery Grant program [RGPIN-2018-04582], the New Frontiers in Research Fund - Exploration [NFRFE-2019-00546], and the Canada Foundation for Innovation John R. Evans Leaders Fund [37871]. Funding support for C. Kelly, R. Habermehl, and J. Fawkes was provided by the TechNation Student Work Placement Program. J. Fawkes was also supported by an NSERC Undergraduate Student Research Award. D. de Ferreyro Monticelli was supported by the Vanier Canada Graduate Scholarship. The authors thank Kevin Tkachuk at Campbell Scientific for helpful conversations on formatting data streams for use in PLUME Dashboard.

Appendix A. Supplementary data

Supplementary material related to this article can be found online at https://doi.org/10.1016/j.envsoft.2022.105600. The Supporting Information is divided into 8 sections. Sections S1.1, S1.2, and S1.3 contain a comprehensive list of the required software and Python packages as well as installation instructions and instructions on how to run the dashboard. Section S2 contains comprehensive documentation on all of the user defined settings, presented in Tables S1 though S16. Section S3 contains an explanation for each of the columns in the Event Markers CSV, presented in Table S17. Section S4 contains comprehensive documentation as well as syntax examples for all of the Event Marker Commands, presented in Table S18. Section S5 contains a template for writing a DAQ script (if the user does not wish to use the provided modbus-tcp_daq.py script). Section S6 provides a discussion on the instrument capacity of the PLUME Dashboard. Section S7 gives a list of text entries (Table S20) that are useful during data analysis. Section S8 provides some figures that clarify how changing the default parameters of the peak detection algorithm would affect the results. Lastly, Authors are currently developing a Q&A to be posted in the GitHub page to clarify common questions about our software.

References

Alavi-Shoshtari, M., Salmond, J.A., Giurcăneanu, C.D., Miskell, G., Weissert, L., Williams, D.E., 2018. Automated data scanning for dense networks of low-cost air quality instruments: Detection and differentiation of instrumental error and local to regional scale environmental abnormalities. Environ. Model. Softw. 101, 34–50.

Badyda, A.J., Grellier, J., Dabrowiecki, P., 2016. Ambient PM2.5 exposure and mortality due to lung cancer and cardiopulmonary diseases in polish cities. In: Respiratory Treatment and Prevention. Springer, pp. 9–17.

Bax, C., Lotesoriere, B.J., Capelli, L., 2021. Real-time monitoring of odour concentration at a landfill fenceline: performance verification in the field. In: NOSE 2021, Vol. 85. Italian Association of Chemical Engineering-AIDIC, pp. 19–24.

Brantley, H., Hagler, G., Kimbrough, E., Williams, R., Mukerjee, S., Neas, L., 2014. Mobile air monitoring data-processing strategies and effects on spatial air pollution trends. Atmos. Meas. Tech. 7 (7), 2169–2183.

Bukowiecki, N., Dommen, J., Prevot, A., Richter, R., Weingartner, E., Baltensperger, U., 2002. A mobile pollutant measurement laboratory—measuring gas phase and aerosol ambient concentrations with high spatial and temporal resolution. Atmos. Environ. 36 (36–37), 5569–5579.

Campailla, A., Giordani, E., 2022. Redis software for windows. accessed on 22-07-2022. URL https://github.com/microsoftarchive/redis/releases.

Canagaratna, M.R., Jayne, J.T., Ghertner, D.A., Herndon, S., Shi, Q., Jimenez, J.L., Silva, P.J., Williams, P., Lanni, T., Drewnick, F., et al., 2004. Chase studies of particulate emissions from in-use New York City vehicles. Aerosol Sci. Technol. 38 (6), 555–573.

Carslaw, D.C., Ropkins, K., 2012. Openair — An R package for air quality data analysis. Environ. Model. Softw. 27–28, 52–61. http://dx.doi.org/10.1016/j.envsoft.2011.09.008.

Chen, Y., Gu, P., Schulte, N., Zhou, X., Mara, S., Croes, B.E., Herner, J.D., Vijayan, A., 2022. A new mobile monitoring approach to characterize community-scale air pollution patterns and identify local high pollution zones. Atmos. Environ. 272, 118936.

Chipkin Automation Systems, 2022. CAS modbus scanner. accessed on 22-07-2022. URL https://store.chipkin.com/products/tools/cas-modbus-scanner.

Collins, G., 2022. pyModbus library for Python. accessed on 22-07-2022. URL https://github.com/riptideio/pymodbus/.

de Medrano, R., de Buen Remiro, V., Aznarte, J.L., 2021. SOCAIRE: Forecasting and monitoring urban air quality in Madrid. Environ. Model. Softw. 143, 105084.

Deshmukh, P., Kimbrough, S., Krabbe, S., Logan, R., Isakov, V., Baldauf, R., 2020. Identifying air pollution source impacts in urban communities using mobile monitoring. Sci. Total Environ. 715, 136979.

Drewnick, F., Böttger, T., Von Der Weiden-Reinmüller, S.-L., Zorn, S., Klimach, T., Schneider, J., Borrmann, S., 2012. Design of a mobile aerosol research laboratory and data processing tools for effective stationary and mobile field measurements. Atmos. Meas. Tech. 5 (6), 1443–1457.

Evans, J., 2022. DB library for Python. accessed on 22-07-2022. URL https://github.com/lgastako/db.

Feenstra, B., Collier-Oxandale, A., Papapostolou, V., Cocker, D., Polidori, A., 2020. The AirSensor open-source R-package and DataViewer web application for interpreting community data collected by low-cost sensor networks. Environ. Model. Softw. 134, 104832.

Gazoni, E., Clark, C., 2022. OpenPyxl library for Python. accessed on 22-07-2022. URL https://github.com/redis/redis-py.

Harris, C.R., Millman, K.J., van der Walt, S.J., Gommers, R., Virtanen, P., Cournapeau, D., Wieser, E., Taylor, J., Berg, S., Smith, N.J., Kern, R., Picus, M., Hoyer, S., van Kerkwijk, M.H., Brett, M., Haldane, A., del Río, J.F., Wiebe, M., Peterson, P., Gérard-Marchant, P., Sheppard, K., Reddy, T., Weckesser, W., Abbasi, H., Gohlke, C., Oliphant, T.E., 2020. Array programming with NumPy. Nature 585 (7825), 357–362. http://dx.doi.org/10.1038/s41586-020-2649-2.

Herndon, S.C., Jayne, J.T., Zahniser, M.S., Worsnop, D.R., Knighton, B., Alwine, E., Lamb, B.K., Zavala, M., Nelson, D.D., McManus, J.B., et al., 2005. Characterization of urban pollutant emission fluxes and ambient concentration distributions using a mobile laboratory with rapid response instrumentation. Faraday Discuss. 130, 327–339.

Ho, W.-C., Hartley, W.R., Myers, L., Lin, M.-H., Lin, Y.-S., Lien, C.-H., Lin, R.-S., 2007. Air pollution, weather, and associated risk factors related to asthma prevalence and attack rate. Environ. Res. 104 (3), 402–409.

Hofman, J., Do, T.H., Qin, X., Bonet, E.R., Philips, W., Deligiannis, N., La Manna, V.P., 2022. Spatiotemporal air quality inference of low-cost sensor data: Evidence from multiple sensor testbeds. Environ. Model. Softw. 149, 105306.

Hossain, S., 2019. Visualization of bioinformatics data with dash bio. In: Calloway, C., Lippa, D., Niederhut, D., Shupe, D. (Eds.), Proceedings of the 18th Python in Science Conference. pp. 126–133. http://dx.doi.org/10.25080/Majora-7ddc1dd1-012.

JET BRAINS, 2022. PyCharm software. accessed on 22-07-2022. URL https://www.jetbrains.com/pycharm/.

Ketzel, M., Berkowicz, R., 2004. Modelling the fate of ultrafine particles from exhaust pipe to rural background: an analysis of time scales for dilution, coagulation and deposition. Atmos. Environ. 38 (17), 2639–2652.

Khedo, K.K., Shenoy, P., Zeadally, K., Awotar, K., Ramdani, L., 2020. A vehicular internet of things (IoT) system for high-granularity air quality monitoring in smart cities. In: Smart Technologies for Smart Cities. Springer, pp. 111–133.

Lefebvre, L., 2022. pyModbusTCP library for Python. accessed on 22-07-2022. URL https://github.com/sourceperl/pyModbusTCP.

Lim, C.C., Kim, H., Vilcassim, M.R., Thurston, G.D., Gordon, T., Chen, L.-C., Lee, K., Heimbinder, M., Kim, S.-Y., 2019. Mapping urban air quality using mobile sampling with low-cost sensors and machine learning in Seoul, South Korea. Environ. Int. 131, 105022.

Liu, B., Zimmerman, N., 2021. Fleet-based vehicle emission factors using low-cost sensors: Case study in parking garages. Transp. Res. D 91, 102635.

Maciejczyk, P.B., Offenberg, J.H., Clemente, J., Blaustein, M., Thurston, G.D., Chen, L.C., 2004. Ambient pollutant concentrations measured by a mobile laboratory in South Bronx, NY. Atmos. Environ. 38 (31), 5283–5294.

Mahajan, S., 2022. Design and development of an open-source framework for citizen-centric environmental monitoring and data analysis. Sci. Rep. 12 (1), 1–14.

Nokeri, T.C., 2022. Dash bootstrap components. In: Web App Development and Real-Time Web Analytics with Python. Springer, pp. 87–97.

Parmer, C., 2022. Dash core components for plotly. accessed on 22-07-2022. URL https://github.com/plotly/dash-core-components.

Plotly Technologies Inc., 2015. Collaborative data science. URL https://plot.ly.

Plotly Technologies Inc., 2022. Dash DAQ. accessed on 22-07-2022. URL https://github.com/plotly/dash-daq.

Redis Inc., 2022. Redis library for Python. accessed on 22-07-2022. URL https://github.com/redis/redis-py.

Tan, Y., Lipsky, E.M., Saleh, R., Robinson, A.L., Presto, A.A., 2014. Characterizing the spatial variation of air pollutants and the contributions of high emitting vehicles in Pittsburgh, PA. Environ. Sci. Technol. 48 (24), 14186–14194.

The pandas development team, 2020. Pandas-dev/pandas: Pandas. http://dx.doi.org/10.5281/zenodo.3509134.

Vagnoli, C., Martelli, F., De Filippis, T., Di Lonardo, S., Gioli, B., Gualtieri, G., Matese, A., Rocchi, L., Toscano, P., Zaldei, A., 2014. The SensorWebBike for air quality monitoring in a smart city. In: IET Conference on Future Intelligent Cities. IET.

Van Rossum, G., Drake, F.L., 2009. Python 3 Reference Manual. CreateSpace, Scotts Valley, CA.

Wallace, J., Corr, D., Deluca, P., Kanaroglou, P., McCarry, B., 2009. Mobile monitoring of air pollution in cities: the case of Hamilton, Ontario, Canada. J. Environ. Monit. 11 (5), 998–1003.

Wang, J.M., Jeong, C.-H., Hilker, N., Shairsingh, K.K., Healy, R.M., Sofowote, U., Debosz, J., Su, Y., McGaughey, M., Doerksen, G., et al., 2018. Near-road air pollutant measurements: accounting for inter-site variability using emission factors. Environ. Sci. Technol. 52 (16), 9495–9504.

Wang, J., Jeong, C.-H., Zimmerman, N., Healy, R., Wang, D., Ke, F., Evans, G.J., 2015. Plume-based analysis of vehicle fleet air pollutant emissions and the contribution from high emitters. Atmos. Meas. Tech. 8 (8), 3263–3275.

Wang, M., Zhu, T., Zheng, J., Zhang, R., Zhang, S., Xie, X., Han, Y., Li, Y., 2009. Use of a mobile laboratory to evaluate changes in on-road air pollutants during the Beijing 2008 Summer Olympics. Atmos. Chem. Phys. 9 (21), 8247–8263.

Weller, Z.D., Yang, D.K., von Fischer, J.C., 2019. An open source algorithm to detect natural gas leaks from mobile methane survey data. PLoS One 14 (2), e0212287.

Xia, T., Raneses, J., Batterman, S., 2022. Improving the performance of pipeline leak detection algorithms for the mobile monitoring of methane leaks. Atmosphere 13 (7), 1043.

Zimmerman, N., 2022. Tutorial: Guidelines for implementing low-cost sensor networks for aerosol monitoring. J. Aerosol Sci. 159, 105872.

Zimmerman, N., Li, H.Z., Ellis, A., Hauryliuk, A., Robinson, E.S., Gu, P., Shah, R.U., Ye, Q., Snell, L., Subramanian, R., et al., 2020. Improving correlations between land use and air pollutant concentrations using wavelet analysis: Insights from a low-cost sensor network. Aerosol Air Qual. Res. 20 (2), 314–328.

Zimmerman, N., Wang, J.M., Jeong, C.-H., Ramos, M., Hilker, N., Healy, R.M., Sabaliauskas, K., Wallace, J.S., Evans, G.J., 2016. Field measurements of gasoline direct injection emission factors: spatial and seasonal variability. Environ. Sci. Technol. 50 (4), 2035–2043.